København Universitets Matematiske Institut
Afdelingen for Informationsbehandling
Universitetsparken 5.

Description of IMP IV

(supplement to IMP III)

by

Peter Lindblad

K.U.A.I.B. 18

Basic Concepts.

The IMP is a means of extending any language with so called macro facilities. The basic concept, the macro, is in its simplest form very similar to the well known use of abbreviations in natural languages e.g. writing IMP instead of Interpretative Macro Processor. To state the fact that a certain string of symbols is used as an abbreviation of (or as we prefer to say, the name of) another string we write a so called macrodefinition which may have the form:

[IMP* Interpretative Macro Processor]

Here IMP is called the macroname and Interpretative Macro Processor the macrostring. As we may want to use the word IMP for its own sake as well as an abbreviation, we use special delimiters to indicate that it is used in the latter sense and write (IMP)
which is called a macrocall.

If the macrostring of a definition does not contain any macrocalls or macrodefinitions the value of the definition is the macrostring itself. If the macrostring contains other definitions these are evaluated and their names and values are placed in a list of currently defined macros. The evaluation takes place in the opposite order of the one in which the definitions appear. If the macrostring contains macrocalls these are replaced by their value in accordance with the latest entry of their names in the list of definitions. When a definition is evaluated all definitions having appeared within its macrostring are removed from the list of definitions.

This method of evaluation implies that the definitions valid when evaluating a certain definition are all definitions

contained in it and all definitions following it, but not those definitions which may appear within some other definition (compare with the concept of locality which applies to variables, procedures etc. in ALGOL).

Substituting the value of a definition for a call is somewhat more complicated when the definitions contain formal parameters, i.e. integers (strings of decimal digits) placed between two =.

If that is the case the macrocall must be supplied with a number of actual parameters not less than the largest integer used as a formal parameter. Actual parameters are macrostrings (these may contain formal parameters) beginning with the delimiter ∗ and placed between the name and the) of the macrocall. When the call is replaced by the value of the definition, each formal parameter is replaced by the corresponding actual parameter (=0= corresponds to the name, =1= to the first actual parameter etc.).

Comments, quotations and delimiter definitions.

Two kinds of comments not having any effect on the result of the processing may be inserted in the input string. One consisting of any string of symbols surrounded by ¢ and ¢ and the other consisting of any string of symbols surrounded by ₹ and ¢. The difference between them is that the latter will be printed out on the monitoring typewriter.

In some cases it may be necessary to include some of the delimiters of the IMP-language in the value of a definition. this may be done by means of the quotation marks ∢ and ∗: every symbol between a ∢ and the first following ∗ will be used as it stands even if it happens to be a delimiter.

The set of delimiters used in the present report need not be the delimiters actually used, as any inputstring must contain a definition of delimiters, and the delimiters may be redefined at any place in the inputstring. A delimiter definition is a string of symbols containing 13 times the symbol ^. The symbols between the first and the second ^ will be used to indicate the beginning of a macrodefinition etc.. The symbols between the 12th and the 13th ^ are dummy symbols which will be ignored in the input. If not otherwise stated in this report it is supposed that the delimiter definitions are

^ [^] ^ (^) ^ * ^ - ^ < ^ > ^ ¢ ^ ! ^ # ^ ^

The symbols that may be used as delimiters are all symbols on 8 channel papertape with uneven parity, except the symbols with binary value 58 and 60 (case shifts), 63 and 127 (all holes), 14 (_or|) and 12 (end code), and these symbols combined with underline or vertical bar. The superdelimiter cannot be used as delimiter ^. If the same symbol appears more than once in a delimiter definition only the first appearance is used. End code symbols will stop the input with the message wait printed out on the monitoring typewriter. When a space is typed input is continued.

The dynamic macros.

The dynamic macros are a set of macros that may be called in the same way as the defined macros, but their values may be taken as references to certain subroutines in the processor rather than strings of symbols.

When these macros are called in a macrostring the corresponding subroutines will be executed every time the processor scans

the value of that macrostring, and the call of the dynamic macro is replaced by the result of this subroutine.

The use of the name dynamic macro is due to the following: When a macro is defined by calls of other defined macros these calls will be replaced by their value during the evaluation of the definition. But a call of a dynamic macro in a macro-definition is not replaced before the said macro is itself called, and the effect of that call may be dependent on the context in which it appears.

The dynamic macros may have side effects on an array of working locations referred to by addresses which are integers, n, in the range $0 \leq n \leq 63$.

In the following description of the dynamic macros the result will not be explicitly mentioned if it is the empty string.

macro name	number of parameters	description
bin	2	will interpret the second parameter as a decimal number and store its binary value in the working location addressed by parameter 1.
cmp	≥ 2	will scan the list of parameters beginning with parameter 3 until one is found, which is equal to parameter 2 or until the last parameter has been used. In working location adressed by parameter 1 will be

		stored -2 plus the number of the first parameter equal to parameter 2 if such a parameter is found and else 0. cmp is an abbreviation of compare.
add	3	will add the contents of working locations adressed by parameters 1 and 2 and store the result in the location addressed by parameter 3.
sub	3	similar to add
mul	3	- - -
div	3	- - -
mod	3	- - -
eq	3	will store 1 or 0 in the location addressed by parameter 3 depending on whether the contents of the locations addressed by parameters 1 and 2 are equal or not.
gr	3	is analogue to the proceeding, but with greater than instead of equal to.
or	3	will store the logical sum of the locations addressed by parameters 1 and 2 in the location addressed by parameter 3.

out	2	will print parameter 2 on the output unit determined by parameter 1.
in	2	the result of this macro is a string read in from the input unit determined by parameter 1. The string must be terminated with the symbol given as parameter 2.
name	0	the result of this macro is the name of the macro definition which is currently being evaluated.
case	≥ 1	the result is determined by the content of the location which is addressed by parameter 1. If this content is n the result is parameter n+2.
dec	1	the result is the decimal representation of the content of the location addressed by parameter 1.
rep	2	if the content of the location addressed by parameter 1 is 0, the result is parameter 2 repeated until the content of the mentioned location becomes 0 (by some sideeffect of the evaluation of parameter 2).

par	1	a call of this macro is equivalent to a formal parameter in which the integer is the content of the location adressed by parameter 1.
call	≥1	the result of this macro is the same as a macrocall in which the name is parameter 1 and the actual parameters are the rest of the parameters.

Working location 0 is used in the following special way: everytime a macro is called, 1 is stored in location 0 if the macrocall has actual parameters, otherwise 0 is stored. When a reference to an actual parameter has been made, 1 is stored in location 0 if the parameter was not the last, otherwise 0 is stored. Location 64 contains the constant 1.

Error messages.

If a syntactical error is found in the input a message will be output on the monitoring typewriter consisting of

1) all left parenthesis to which no corresponding right parenthesis has been found,
2) the erroneous symbol (in red)
3) the number of the line in which the erroneous symbol appeared.

The erroneous symbol is then treated as if it had been quoted, and the processor continues. If the erroneous symbol is ¢ or) no message is printed.

During the evaluation of a definition four kinds of errors are recognised: missing actual parameter, call of not defined macro, missing actual parameter to a dynamic macro, and a dynamic macro tries to store a value outside the working locations 0-63. When one of these errors occur the processor will proceed as follows: store a number describing the error (0, 1, 2 or 3) in location 65, store the number of the line, in which the macrocall that caused error appeared, in location 66, and then call the macro with the empty string as name. This macro is predefined as follows, but may be redefined.

```
[* (out*17*
   (case *65* -par * -def * -arg * index) =0= (dec *66*))]
```

When this call is finished the processor will continue.

If the capacity of the corestore or the drum is not sufficient the processor will terminate with the messages alas or drum.